Modelagem Específica de Domínio em Linhas de Produto de Software na Computação Ubíqua

Eduardo F. Z. Santana, Raphael P. de Oliveira, Antonio F. do Prado, Wanderley L. de Souza, Mauro Biajiz

Departamento de Computação (DC) – Grupo de Computação Ubíqua (GCU)
Universidade Federal de São Carlos (UFSCar)
Caixa Postal 13.565-905 – São Carlos – SP – Brasil

{eduardo, raphael_oliveira, prado, desouza, mauro}@dc.ufscar.br

Abstract. *This paper presents an approach for software development based on Domain-Specific Modeling (DSM) of Software Product Lines (SPL) in ubiquitous computing. Focused on the problem domain is modeled the product line and based on the DSM is produced a metamodel that supports the development of various products. A framework, called Ubiquitous Computing Framework (UCF) that meets non-functional requirements of ubiquitous computing is one of the core assets of product lines. The tool MVCASE and a code generator automate most of the activities of the proposed approach.*

Resumo. *Este artigo apresenta uma abordagem para o desenvolvimento de software, baseado na Domain-Specific Modeling (DSM), de Linhas de Produtos de Software na computação ubíqua. Com foco no domínio do problema modela-se a Linha de Produtos, e baseado na DSM, elabora-se um metamodelo que suporta a construção de diversos produtos. Um framework, denominado Ubiquitous Computing Framework (UCF) que atende requisitos não funcionais da Computação Ubíqua, é um dos assets do núcleo da Linha de Produtos. A ferramenta MVCASE e um gerador de código automatizam grande parte das atividades da abordagem proposta.*

1. Introdução

O termo Computação Ubíqua refere-se a ambientes saturados de dispositivos computacionais e redes de comunicação, que se integram naturalmente à atividade humana. Segundo [Weiser 1994], “as mais profundas tecnologias são as que desaparecem”. Neste sentido a Computação Ubíqua pode ser considerada o oposto da Realidade Virtual. Enquanto na segunda o usuário penetra no mundo virtual, na primeira é a computação que penetra no mundo físico do usuário, construindo a ligação entre os dois mundos.

Com o avanço da capacidade de hardware e de tecnologias chaves de software e de redes, a Computação Ubíqua está se tornando uma realidade [Pham et al. 2007]. Entretanto o desenvolvimento de software para este novo paradigma, ainda possui diversas dificuldades, como: desenvolvimento de forma manual e sob demanda; falta de capacidade de expansão; e dificuldade de manutenção e evolução [Helal 2005].

Tais dificuldades podem ser observadas pela falta de metodologias, processos, técnicas e ferramentas para desenvolvimento, implementação, teste, e manutenção

[Spínola et al. 2007] de software na computação ubíqua. Motivados em diminuir essas dificuldades, esse artigo apresenta os resultados de uma pesquisa que investiga uma abordagem para o desenvolvimento de software, baseado na Modelagem especifica de domínio (*Domain-Specific Modeling* - DSM), de Linhas de Produtos de Software (LPS) na Computação Ubíqua. A abordagem visa facilitar o desenvolvimento de software na computação ubíqua, através de uma abordagem com uma arquitetura bem definida e o reúso em aplicações ubíquas.

Na DSM [Kelly e Tolvanen 2008] desenvolve-se um metamodelo que, representa o conhecimento comum das aplicações desse domínio, e posteriormente, facilitada pelo reúso desse metamodelo, desenvolvem-se as aplicações. Em uma LPS [Clements e Northrop, 2001] são construídos softwares que possuem funcionalidades comuns a todos os produtos e especificas dependentes de cada produto. A especificação dessas funcionalidades é realizada conforme o domínio no qual a LPS se enquadra. Uma modelagem especifica, direcionada cada domínio, pode proporcionar maior grau de reúso, do que os processos que empregam apenas componentes e *frameworks*.

A seqüência deste artigo está assim organizada: a seção 2 introduz os principais conceitos e técnicas utilizadas na abordagem; a seção 3 apresenta a abordagem proposta; a seção 4 apresenta a Engenharia de Domínio; a seção 5 apresenta a instanciação de produtos pela Engenharia de Aplicação; a seção 6 discute os trabalhos correlatos; e finalmente a seção 7 apresenta as conclusões relativas a esse trabalho, apontando direções para trabalhos futuros.

2. Principais Conceitos e Técnicas da Abordagem Proposta

Nesta seção são apresentados as principais técnicas e conceitos utilizados para o desenvolvimento da abordagem proposta. A seção 2.1 apresenta a DSM. A seção 2.2 discorre sobre LPS, e a seção 2.3 apresenta o *Ubiquitous Computing Framework* (UCF) [Santana et. al. 2007], um *framework* de componentes reutilizado na construção de diferentes produtos da LPS.

2.1. Modelagem Específica de Domínio (DSM)

Na *Model-Driven Architecture* (MDA) [OMG 2008] as especificações e funcionalidades do software são modeladas através de um modelo independente de plataforma (*Platform-Independent Model* - PIM), enquanto que as tecnologias e plataformas são modeladas através de um modelo especifico de plataforma (*Platform-Specific Model* - PSM). Usando mecanismos de transformação, a partir de um PIM, vários PSMs podem ser obtidos e a partir de cada PSM, podem ser construídos geradores de código para diversas plataformas.

O processo proposto para a MDA é conhecido como *Model-Driven Development* (MDD) [Cicchetti et al. 2007], e os seus principais artefatos gerados são os modelos, e não o código fonte. A partir desses modelos, com o uso de geradores de códigos e *frameworks*, é possível gerar todo ou grande parte do código de uma aplicação, necessitando de pouca ou nenhuma alteração manual para se obter o código executável.

Com foco nas idéias da MDA e a MDD, tem-se a DSM que utiliza uma metalinguagem, com seu metamodelo para representar cada domínio. Na DSM, para cada domínio é definida uma metalinguagem, diferindo de outras metalinguagens mais

genéricas, como a *Unified Modeling Language* (UML) que visa representar o paradigma Orientado a Objetos. Na DSM, os modelos são mais específicos e completos, e recursos como *frameworks*, padrões de projeto e componentes são incluídos na modelagem com o objetivo de gerar maior quantidade de código e com melhor qualidade [Greenfield e Short 2004].

2.2. Linhas de Produtos de Software

Uma LPS visa o desenvolvimento em larga escala, onde produtos são obtidos a partir do reúso de *assets* previamente construídos e disponíveis em um repositório, na forma de componentes, padrões ou frameworks. O *core asset* da LPS é composto por funcionalidades comuns e opcionais. As funcionalidades comuns constituem o *kernel* da LPS e são encontradas em todos os produtos da LPS. As funcionalidades opcionais são as possíveis extensões do *core asset* que podem ser reutilizados por alguns produtos da linha, mas não necessariamente por todos.

Conforme Figura 1, produtos da LPS são construídos a partir do Kernel e das especificações das funcionalidades opcionais (*Optional*) definidas no *core asset*, complementadas com o desenvolvimento de partes específicas (*Application*) para um determinado produto.

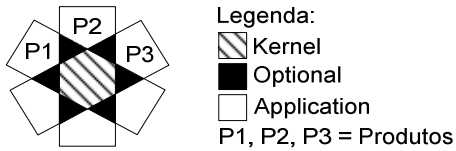

Figura 1. LPS e seus Produtos.

O processo de desenvolvimento da abordagem proposta combina técnicas de LPS e de DSM para elaborar o metamodelo do domínio do problema que suporte tanto a modelagem, como a implementação de diferentes produtos de uma LPS. Na Computação Ubíqua, onde se tem uma grande diversidade de dispositivos e contextos, um mesmo produto precisa ser implementado em diferentes arquiteturas. Portanto, no desenvolvimento de uma LPS na Computação Ubíqua, torna-se fundamental o uso da DSM, possibilitando a modelagem dos produtos e suas implementações nas diferentes plataformas encontradas na Computação Ubíqua.

2.3. Ubiquitous Computing Framework (UCF)

O *Ubiquitous Computing Framework* (UCF) foi desenvolvido no Departamento de Computação da UFSCar [Santana et al. 2007], para atender requisitos relacionados com: (a) ciência de contexto, para possibilitar o uso das informações contextuais relativas ao ambiente computacional, pessoal e físico, na oferta de conteúdos e serviços customizados; (b) adaptação de conteúdo, para suportar o acesso ubíquo às aplicações, ou seja, para que essas aplicações possam ser acessadas de qualquer lugar, a qualquer hora e usando qualquer tipo de dispositivo [Fallis et al. 2007]; (c) descoberta onipresença e composição de serviços web semânticos, para permitir às aplicações o uso pleno dos serviços web disponíveis.

Considerando que o desenvolvimento de uma aplicação ubíqua geralmente envolve duas partes: cliente e servidor, o UCF foi organizado em dois grandes pacotes UCFClient e UCFServer, conforme mostra o modelo de componentes da Figura 2.

O projeto da parte do cliente utiliza componentes dos pacotes *UCFClient.* O pacote *UCFClient* provê os seguintes componentes: *UserInterface* é responsável pela apresentação das informações acessadas via o dispositivo móvel; *ContextAgentClient* monitora as informações relevantes ao contexto de uso da aplicação, enviando-as periodicamente ao servidor; *OfflineRecords* armazena informações do cliente e do servidor, disponibilizando-as em caso de falta de conectividade; *CalloutProtocolClient* é responsável pela comunicação entre cliente e servidor; e *P2PProtocolClient* é responsável pela comunicação entre dois clientes.

O projeto da parte do servidor utiliza componentes dos pacotes *UCFServer*. O pacote *UCFServer* provê os seguintes componentes:. *ServerManager* gerencia o comportamento do servidor; *CalloutProtocolServer* permite a comunicação entre o tracker e os pares; *AgentFactory* instancia os agentes de software; *ServicesAgent* busca, compõe e monitora os serviços web semânticos; *MobilityManager* gerencia a mobilidade dos agentes de software; *AdaptationAgent* adapta conteúdos acessados via os clientes; *Cache* armazena os conteúdos mais acessados da rede, melhorando dessa forma o desempenho do servidor; *ContextAgentServer* atualiza informações contextuais; *ContextRepository* armazena informações contextuais; e *BrokerAgent* usa ontologias para definir necessidades do usuário.

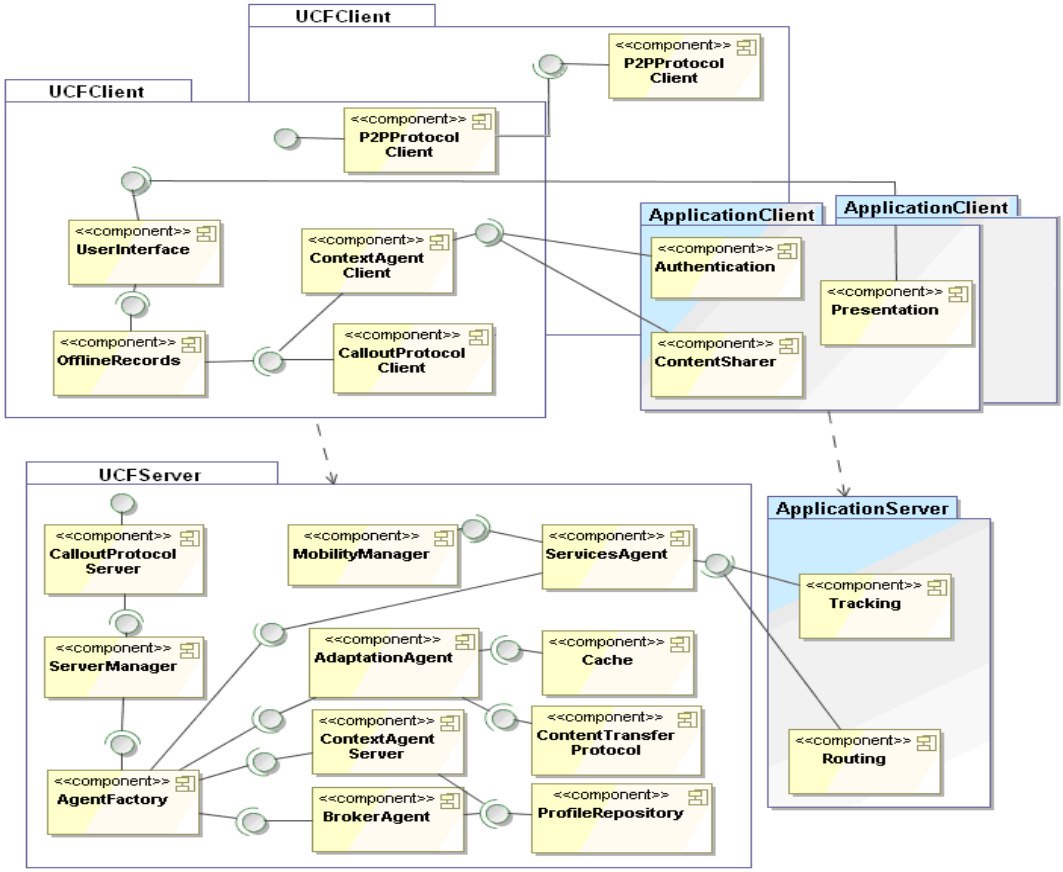

Figura 2. Modelo de componentes de uma aplicação com reúso do UCF.

Conforme ilustra a Figura 2, na fase de projeto do processo de desenvolvimento de aplicações ubíquas modelam-se os componentes da aplicação, conectando-os aos

componentes do UCF. Na implementação das aplicações o UCF é reutilizado importando a biblioteca que contem sua implementação.

3. Abordagem Proposta

Tendo como base o trabalho de Oliveira [Oliveira et al. 2009], propõe-se uma abordagem para o desenvolvimento de software na computação ubíqua, baseado na modelagem específica de domínio em linhas de produto de software, a qual é realizada em duas etapas: Engenharia de Domínio (ED) e de Aplicação (EA). Conforme mostra a Figura 3, a partir dos requisitos do domínio do problema, e orientado por técnicas da DSM e LPS, elabora-se um metamodelo com recursos gráficos e textuais, que suporta a construção de modelos de diferentes produtos desse domínio. Com esse metamodelo é possível reutilizar o conhecimento do domínio do problema e automatizar grande parte do desenvolvimento dos produtos da LPS. Os principais mecanismos para apoiar os Engenheiros de Domínio e de Aplicação são uma ferramenta CASE, um gerador de código, e a *Integrated Development Environment* (IDE) Eclipse [Eclipse 2009].

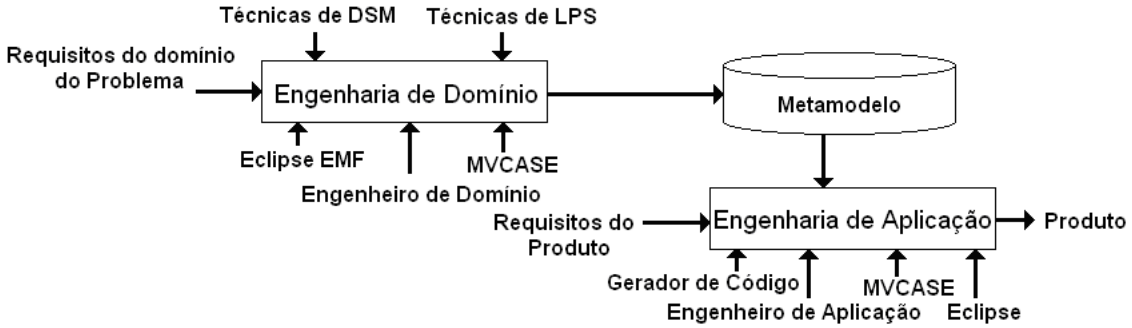

Figura 3. Processo de Desenvolvimento da Abordagem Proposta.

Processos baseados na DSM vêm sendo investigados em diversos domínios, como Educação e Saúde, com casos de sucesso [Malek et al. 2008] [Hawryszkiewycz 2007]. No Departamento de Computação da UFSCar, o grupo de computação ubíqua vem pesquisando essa abordagem, tendo como foco o domínio de aplicações ubíquas na área de ensino na medicina, particularmente no domínio de Portfólio Reflexivo Eletrônico (PRE). Ressalte-se que a abordagem, ao combinar as idéias da DSM e LPS, às vezes emprega o termo **Aplicação** com o mesmo significado de **Produto**.

4. Engenharia de domínio

A ED inicia com os requisitos do domínio do problema, identificando as *features* da LPS, que dão origem aos *assets* que serão reutilizados na EA.

Na ED, o metamodelo inclui todas as *features* que darão origem aos *assets* que constituirão o *core asset* da LPS. Na EA são construídos os produtos da LPS instanciando o metamodelo com o reúso do *core asset*. Tanto na ED como na EA o desenvolvimento segue as disciplinas de Requisitos, Projeto e Implementação, conhecidas dos processos de desenvolvimento de software, e modificadas para atender as necessidades da DSM, LPS e da Computação Ubíqua.

4.1. Requisitos

Nessa disciplina os requisitos do domínio do problema são elicitados, especificados, analisados e verificados. Nas especificações os requisitos são representados em modelos gráficos e textuais, independentes de plataforma. Uma das técnicas utilizadas para especificar os requisitos, nessa etapa, é o diagrama de casos de uso, que é a base para a identificação de uma *feature* da LPS. Uma *feature*, conforme [Chastek et al. 2001], é "um aspecto importante visível ao usuário, de boa qualidade, ou característica de um sistema de software ou sistemas de softwares", e pode representar um ou mais casos de uso. Um exemplo é a *feature* Gerenciar Atividade, que representa os casos de uso: "Gerenciar Reunião", "Gerenciar Pequeno Grupo" e "Gerenciar Prática", conforme mostra o modelo da Figura 4. Nesse modelo, o estereótipo <<Kernel>> representa funcionalidades que serão comuns a todos os produtos da LPS e o estereótipo <<Optional>> representa as funcionalidades opcionais.

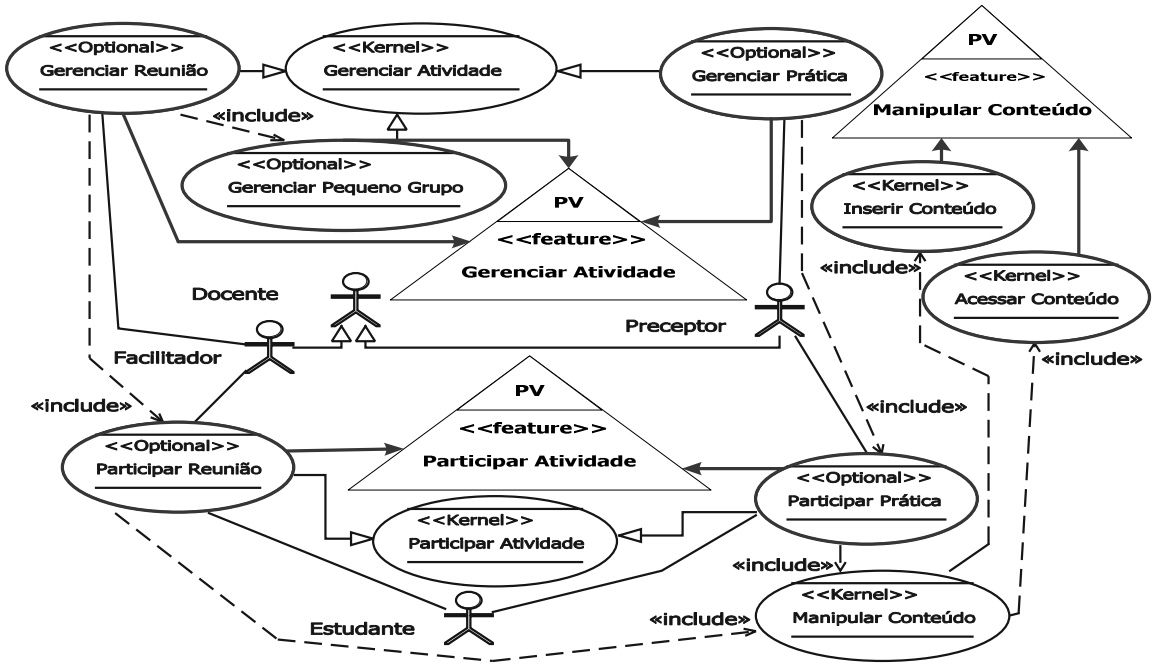

Figura 4. Casos de Uso e Features

Em seguida, as *features* são modeladas com seus relacionamentos, fornecendo uma representação das funcionalidades da LPS. Por exemplo, no modelo da Figura 5, têm-se *features* do domínio, com destaque para as *features* (sombreadas) relacionadas com a Adaptação de Conteúdo e Ciência de Contexto (UCF). Portanto, além dos requisitos funcionais do domínio PRE, foram considerados também os requisitos não funcionais da Adaptação de Conteúdo e Ciência de Contexto das aplicações ubíquas. Nesse modelo, PV representam *features* que são pontos variáveis. O círculo preenchido indica um relacionamento que associa uma *feature* obrigatória ou variável, e o círculo vazio associa uma *feature* opcional ou variável. Por exemplo, nesse modelo, as *features* "Inserir Conteúdo", "Acessar Conteúdo" e "Descobrir Contexto" são obrigatórias para todos os produtos da LPS, e a *feature* "Adaptar Conteúdo" é opcional, dependendo do produto.

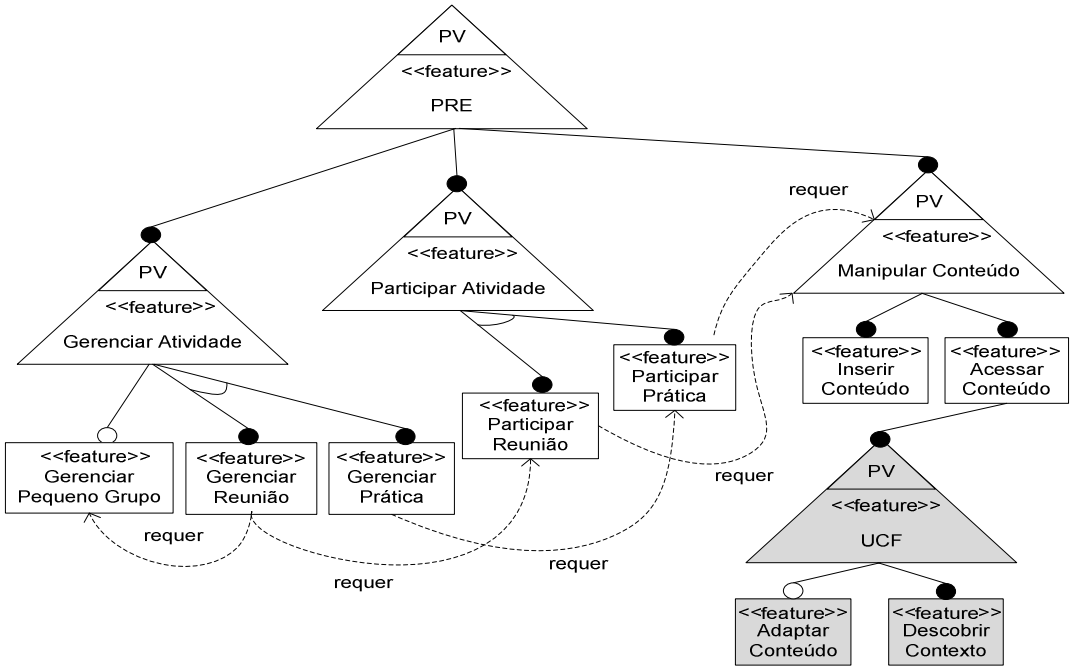

Figura 5. Modelo de Features.

Ainda neste passo da abordagem, as *features* são relacionadas com os possíveis produtos da LPS, conforme Tabela 1.

Tabela 1. Mapeamento de *features* em Produtos.

Features \ Produtos			LaptopPRE	CellPRE
UCF	Adaptar Conteúdo			V
	Descobrir Contexto		V	V
PRE	Participar Atividade	Participar Reunião	V	
		Participar Prática		V
	Manipular Conteúdo	Inserir Conteúdo	V	V
		Acessar Conteúdo	V	V
	Gerenciar Atividade	Gerenciar Reunião	V	
		Gerenciar Prática		V
		Gerenciar Pequeno Grupo	V	

4.2. Análise

Baseado nos modelos de *features* o Engenheiro de Domínio especifica os modelos de classes, com seus atributos, comportamentos e relacionamentos, conforme mostra a Figura 6. Esses modelos, ainda independentes de plataforma (PIM), são refinados com as informações que descrevem com mais detalhes as *features* em cada domínio.

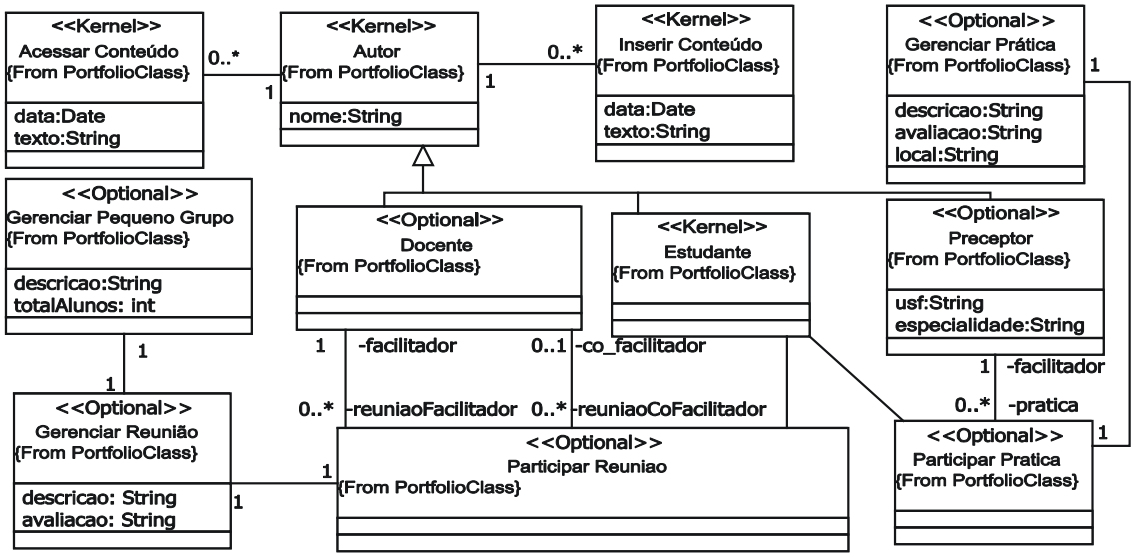

Figura 6. PIM do PRE.

4.3. Projeto do metamodelo

As decisões de projeto, adotando plataformas de hardware e software da LPS, como Java (JEE e JME), e outras decisões, possibilitam refinar os PIMs, obtidos na Análise, conforme as idéias da Modelagem Específica do Domínio. A Figura 7, por exemplo, mostra um Modelo Especifico de Plataforma (PSM) do PRE. Os estereótipos *Kernel* e *Optional* indicam classes obrigatórias e opcionais para construção dos produtos da LPS.

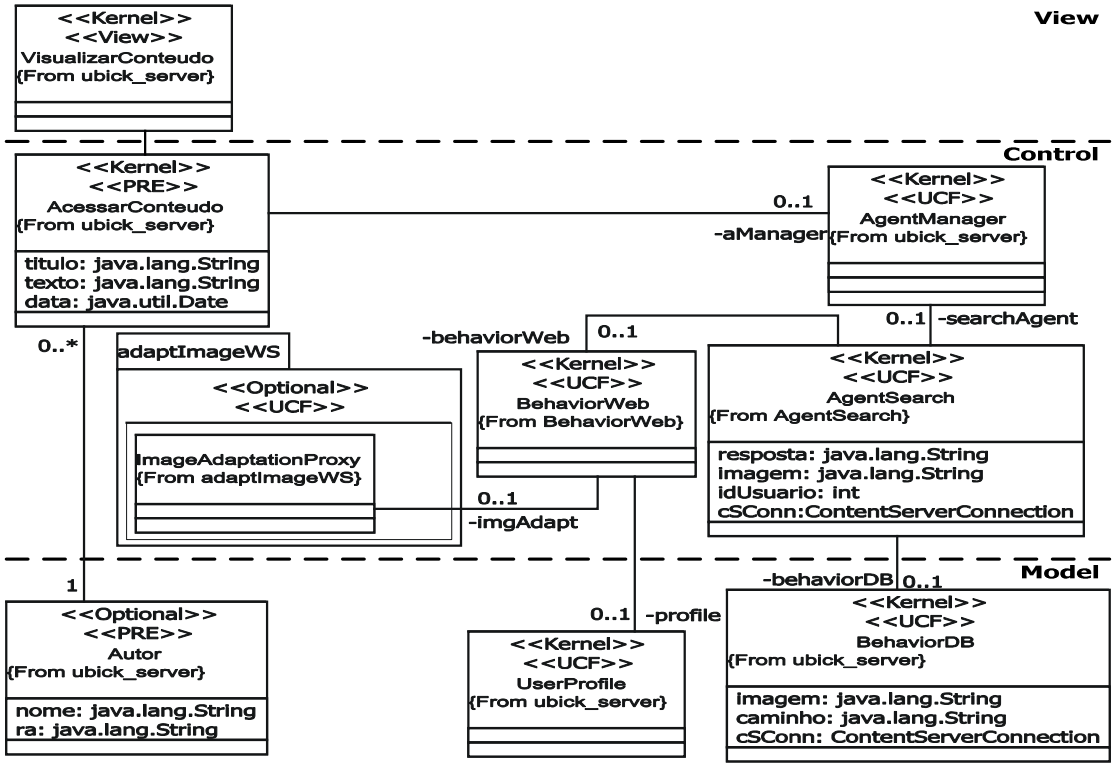

Figura 7. PSM da Linha de Produto de Software PRE.

Prosseguindo, nessa fase da abordagem, com base nos Modelos Específicos de Plataforma (PSMs), elabora-se o metamodelo da LPS, no caso do domínio PRE. Conforme as idéias da DSM, o metamodelo deve possibilitar a construção dos produtos da LPS, considerando as diferentes arquiteturas da Computação Ubíqua, provenientes do uso de diferentes dispositivos móveis (celulares, PDAs e outros), das preferências dos usuários, e das características de redes e conteúdos.

O metamodelo foi especificado com o *framework* EMF [Eclipse EMF 2009], e em uma arquitetura Cliente e Servidor, para possibilitar que parte da aplicação seja executada num servidor e parte seja executada no dispositivo móvel, um requisito da computação ubíqua.

A Figura 8 mostra parte do metamodelo construído do lado servidor da aplicação. As classes desse metamodelo foram obtidas a partir dos PSMs, como por exemplo, as classes *ImageAdaptationProxy*, *BehaviorWeb* e *UserProfile*, extraídas do PSM apresentado na Figura 7. O metamodelo inclui também classes para tratar a adaptação de conteúdo e ciência de contexto das aplicações ubíquas como: *Server* que representa o servidor onde o serviço está disponível; WS que representa os serviços web disponíveis num *proxy*; *Service* que representa os serviços web; *Method* que representa os métodos dos serviços; *Parameter* que representa os parâmetros de um método; *ContextParameter* que representa os parâmetros de contexto; e *Ontology* que representa as ontologias das aplicações ubíquas.

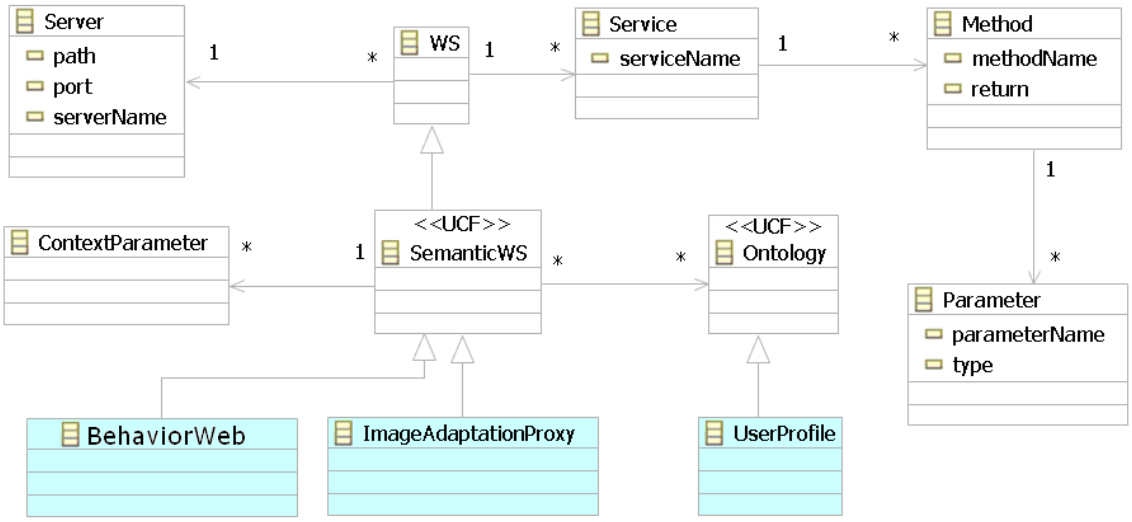

Figura 8. Metamodelo para modelagem de produtos no lado Servidor.

A Figura 9 mostra parte do metamodelo do lado cliente (dispositivo móvel) da aplicação. As classes desse metamodelo também foram obtidas dos PSMs, como por exemplo as classes *AcessarConteudo* e *VisualizarConteudo* extraídas do PSM apresentado na Figura 7. O metamodelo deve atender os diferentes requisitos funcionais e não funcionais da LPS. Assim por exemplo, têm-se as classes comuns das aplicações clientes como: *Task*, identificada pelo seu nome (*taskName*), que representa as ações possíveis de serem realizadas pelo usuário da aplicação; *Service* que deriva de *Task* e representa as ações onde existe pouca interação com o usuário, ou serviços de *background*; *Activity*, que também deriva de *Task* e representa as ações onde existem

interações com usuário; *View* que representa as interfaces de usuário e está sempre relacionada a uma *Activity* com seus atributos tela *touch screen* (*touch*) e titulo (*title*); *ContentStorage* responsável pela persistência de dados; *Database* que deriva de *ContentStorage* e representa a persistência em um banco de dados; *File* que também deriva de ContentStorage e representa a persistência em arquivos; *InternetConnection* que representa conexão com a Internet; GPRS que deriva de InternetConnection e representa conexões GPRS; P2PConnection (Peer-to-Peer) que também deriva de InternetConnection e representa conexões peer-to-peer; Bluetooth que deriva de P2PConnection e representa conexões Bluetooth; Wi-Fi que estende os dois tipos de conexões via protocolo Wi-FI; ServiceCall que representa a chamada a um serviço disponível na Internet ou em outro dispositivo (e.g. serviço de adaptação de conteúdo); e *Sensor* que representa os sensores usados em uma aplicação, e que devem estar disponíveis no dispositivo.

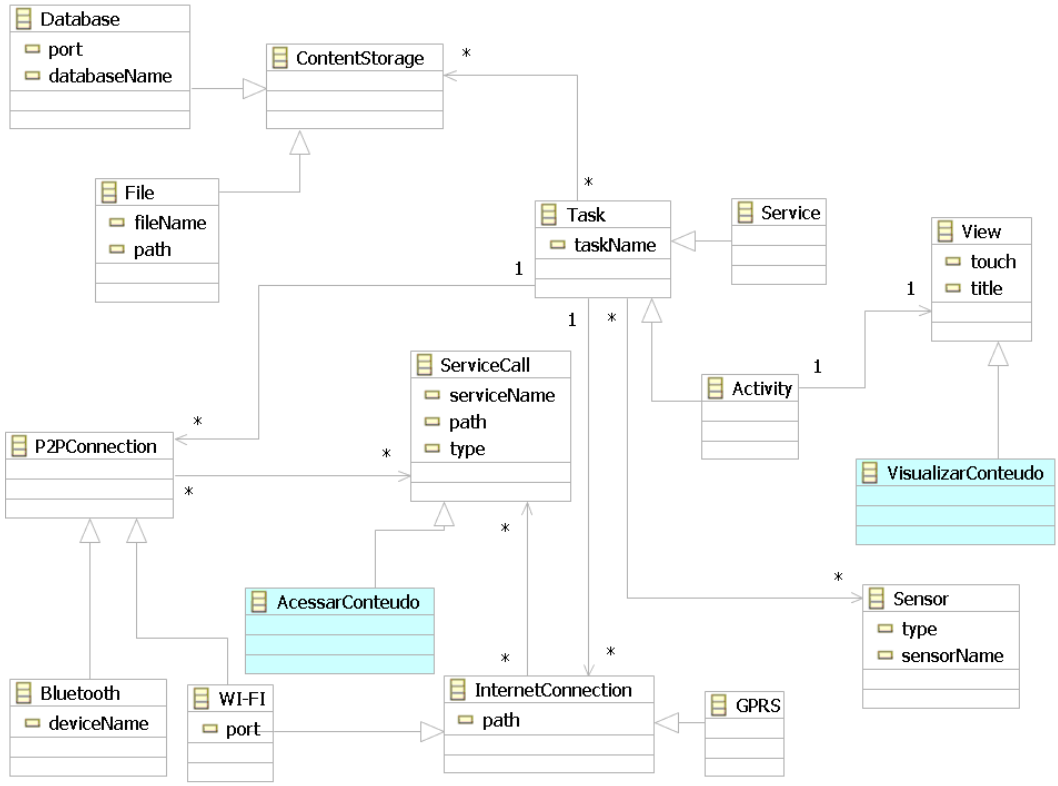

Figura 9. Metamodelo para modelagem da Aplicação no lado Cliente.

Na elaboração do metamodelo, o Engenheiro de Domínio deve anotar com estereótipos as classes para orientar a reutilização e implementação na construção dos produtos. Por exemplo, classes que tratam os aspectos de adaptação de conteúdo e ciência de contexto, passam a ser de responsabilidade do *framework* UCF, um dos *assets* da LPS.

5. Engenharia de Aplicação

Na EA, os produtos da LPS são desenvolvidos, com base no metamodelo construído. Para apoiar o Engenheiro da Aplicação na abordagem proposta, o metamodelo e um gerador de código para a plataforma JME foram integrados na *Multiple-View CASE*

(MVCASE). Dessa forma, consegue-se automatizar e tornar mais ágil grande parte das tarefas do Engenheiro da Aplicação.

A LPS no domínio PRE possibilitou o desenvolvimento dos produtos *LaptopPRE* e *CellPhonePRE*. O *LaptopPRE* visa apoiar as atividades do processo de aprendizagem realizadas dentro da Universidade. O *CellPhonePRE* apóia as atividades realizadas em Unidades de Saúde da Família e na comunidade.

5.1. Modelagem

Nessa fase modela-se o produto como instância do metamodelo. A Figura 10 mostra, por exemplo, um modelo de classes da parte móvel do produto *CellPhonePRE*. Nesse modelo, os estereótipos *Activity* e *View*, *Service*, *ServiceCall*, *VisualizarConteudo* e *AcessarConteudo* identificam as classes instanciadas do metamodelo da LPS, e o estereótipo *Application* identifica as classes específicas do produto.

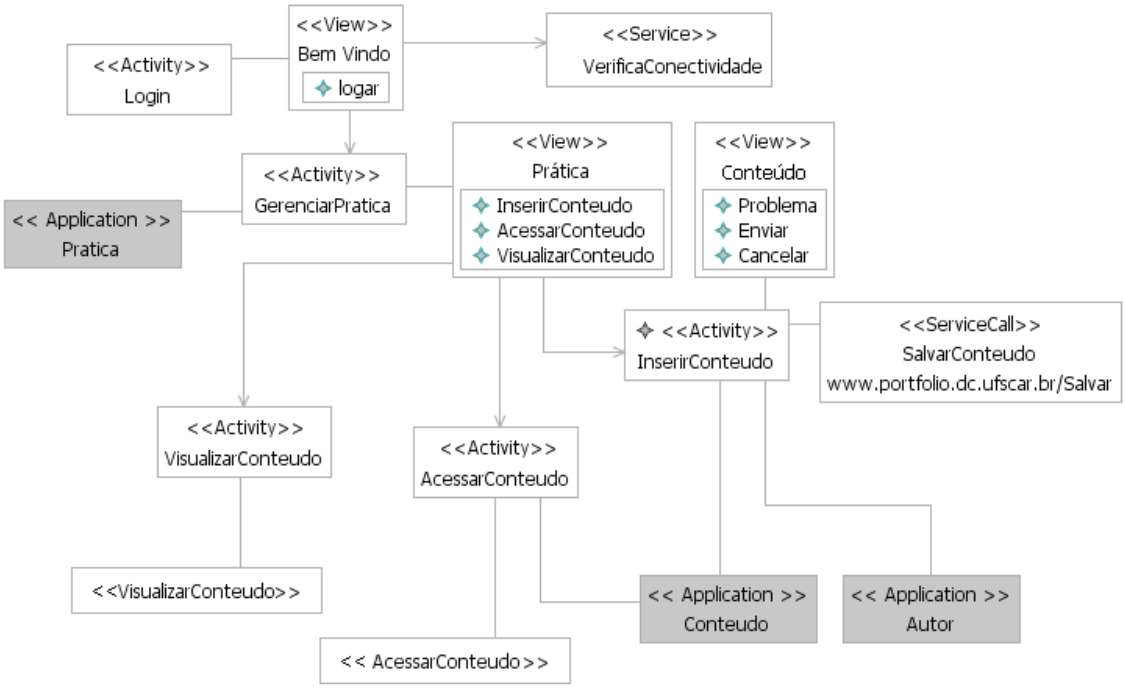

Figura 10. Modelos de Classes do *CellPhonePRE*.

5.2. Implementação

Na MVCASE, com base no modelo de classes, especificados na etapa de modelagem, são gerados os códigos parciais do produto. O gerador de código baseia-se nos estereótipos para evitar repetições e redundâncias na implementação do produto. Por exemplo, o código das classes com o estereótipo UCF é provido pelo *framework* UCF, disponível como um *core asset* da LPS. Na IDE Eclipse, esse código é complementado pelo Engenheiro da Aplicação, até uma versão executável do produto, conforme a Figura 11.

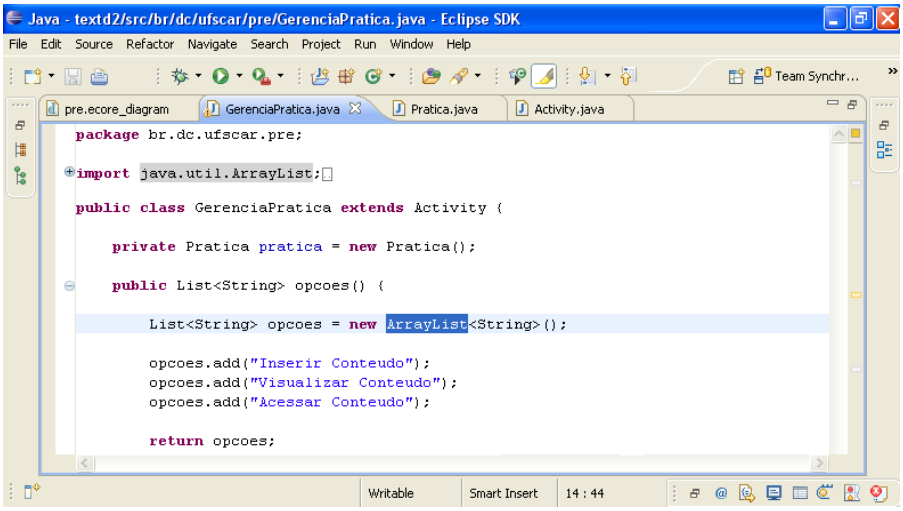

Figura 11. Implementação do produto da LPS.

Finalmente, o produto pode ser executado, conforme mostram as telas da Figura 12, do *LaptopPRE* e *CellPhonePRE*. Os testes com a execução do produto fornecem o *feedback* para retorno ou não aos passos anteriores da abordagem.

Figura 12. *LaptopPRE* e *CellPhonePRE*.

6. Trabalhos Relacionados

Existem diferentes propostas para o desenvolvimento de aplicações ubíquas baseadas no reúso, que utilizam *frameworks*, *middlewares*, ferramentas, modelos de programação, metodologias e padrões de projeto. Contudo, a proposta neste artigo difere dessas propostas por combinar idéias reúso de LPS e da DSM.

Em [Woojin et al. 2007] é proposta uma abordagem para o desenvolvimento *middlewares* em redes de sensores utilizando LPS. Embora os passos utilizados nesse trabalho para o desenvolvimento do *core asset* e seus produtos sejam semelhantes ao da abordagem apresentada neste artigo, o mesmo não é baseado na DSM.

Os trabalhos [Carton et al. 2007] e [Fernandes et al. 2007] empregam a MDA na Computação Ubíqua. A MDA é usada para facilitar o desenvolvimento de aplicações independentes de plataforma, uma vez que os requisitos da Computação Ubíqua podem ser altamente mutáveis. A abordagem, proposta neste artigo, difere dessas abordagens por ser orientada para LPS, o que possibilita a criação de uma família de produtos derivados de um mesmo núcleo, mas que sejam específicos para os diferentes dispositivos usados numa aplicação ubíqua.

Em [Bragança e Machado 2007] apresenta-se o uso da MDA na construção de uma LPS usando ferramentas baseadas no EMF e UML. Em comparação a esse trabalho, a LPS proposta tem a vantagem do uso da ferramenta MVCASE em conjunto com o framework UCF, facilitando o reúso de software no domínio da Computação Ubíqua.

7. Conclusões

Ao explorar as idéias, técnicas e apoios computacionais para a abordagem de LPS, baseada na DSM, no domínio de aplicações ubíquas, este trabalho contribui para a Engenharia de Software nas seguintes áreas:

- Reúso de Software, considerando que tem foco em LPS, baseia-se na DSM, e torna disponível um metamodelo com *core assets* para reúso na construção dos produtos;

- Linha de Produto de Software. Uma LPS para aplicações ubíquas no domínio PRE é construída, fornecendo *assets* para construção de seus produtos. As funcionalidades do núcleo da LPS foram definidas para atender grande parte dos requisitos da Computação Ubíqua, relacionados com a adaptação de conteúdo e ciência de contexto;

- Modelagem Específica de Domínio (DSM). Um metamodelo da LPS é construído, possibilitando o desenvolvimento de seus produtos, com a geração parcial do código. A grande vantagem da DSM é o reúso no nível de modelagem. Uma mesma aplicação pode ser implementada em diferentes dispositivos da computação ubíqua;

- Apoio Computacional. A ferramenta MVCASE e um gerador de código foram integrados com a IDE Eclipse para suportar grande parte das atividades dos Engenheiros de Domínio e de Aplicação.

Concluindo, reforça-se que o reúso através de LPS, baseada na DSM, pode auxiliar o Engenheiro de Software reduzindo o tempo de desenvolvimento das aplicações, diminuindo seus custos e facilitando suas manutenções futuras. Este trabalho prossegue com pesquisas em outros domínios além de Portfólios Reflexivos Eletrônicos, e também visando melhorias e refinamentos no suporte computacional, e no processo da abordagem, para automatizar ainda mais as transformações dos modelos PIM e PSM e possibilitar a geração de código para outras plataformas.

Referências

Carton, A., Clarke, S., Senart, A. e Cahill, V. (2007) “Aspect-Oriented Model-Driven Development for Mobile Context-Aware Computing”. Anais da International Conference on Software Engineering Workshops, p. 5-5.

Chastek, G., Donohoe, P., Kang, K. C. e Theil, S. (2001) “Product Line Analysis: A Practical Introduction” (CMU/SEI-2001-TR-001, ADA396137). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Cicchetti, A., Ruscio D. Di, Salle, A. Di. (2007) Software customization in model driven development of web applications. In: Symposium on Applied Computing Proceedings of the 2007 ACM symposium on applied computing.

Clements, P. Northrop, L.M. “Software Product Lines: Practices and Patterns”, *Addison Wesley*, August, 2001, p. 608.

Eclipse (2009), Disponível em http://www.eclipse.org, último acesso em maio de 2009.

Eclipse EMF (2009). “Eclipse Modeling Framework”. Disponível em http://www.eclipse.org/emf, último acesso em maio de 2009.

Fallis, S., Payne, R., Limb, R. e Allison D. (2007) “Pervasive information, the key to ‘true mobility”, BT Technology Journal, vol. 25, no. 2.

Greenfield, J.; Short, K. Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools; In: Third International Conference, SPLC 2004, Boston, USA, 2004.

Hawryszkiewycz, (2007). I. T. Providing agent support for collaborative systems: using a domain-oriented design method. In: International Journal of Agent-Oriented Software Engineering, V1, p. 175 – 192

Helal, S. (2005) “Programming Pervasive Spaces”, Pervasive Computing, vol. 1, no. 1, pp. 84-87.0

Kelly, S., Tolvanen, JP.: (2008) Domain-Specific Modeling: Enabling full code generation, John Wiley & Sons, ISBN 9780047003666, 427 p.

Malek, J., Laroussi, M., and Derycke, A (2008): ContAct-Us: a context-activity adaptive modeler for ubiquitous learning systems. In Proceedings of the 5th international Conference on Soft Computing As Transdisciplinary Science and Technology. CSTST '08. NY, 530-535

Oliveira, R.P., Prado, A.F, Souza, W. L. e Biajiz, M. (2009) “Development based on MDA, of Ubiquitous Applications Domain Product Lines”. A ser Publicado no 8th IEEE/ACIS International Conference on Computer and Information Science.

OMG (2008). “Model Driven Architecture”. Disponível em http://www.omg.org/mda/ , último acesso em abril de 2009.

Pham, N., Mahmoud, H., Ferworn, A. e Sadeghian, A. (2007) “Applying Model-Driven Development to Pervasive System Engineering”. Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments, pp. 7.

Santana, L.H.Z., Prado, A.F., Souza, W. L. e Biajiz, M. (2007) “Usando Ontologias, Serviços Web Semânticos e Agentes Móveis no Desenvolvimento Baseado em Componentes”. Publicado no Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software, Campinas, v. 1. p. 163-176.

Spínola, R., Massollar, J., Travassos, G.: (2007) Checklist to Characterize Ubiquitous Software Projects. Anais do Sim, Brasileiro de Engenharia de Software, pp. 39-55.

Weiser M. (1994), “The world is not a desktop” ACM Interactions vol. 1, no.1, pp. 7-8.

Woojin, L., Sungwon, K. e Hyung, L. D. (2007) “Product Line Approach to Role-Based Middleware Development for Ubiquitous Sensor Network”. Computer and Information Technology. 7th IEEE International Conference on 16-19 Oct., p. 1032 – 1037.